Chai et al. EURASIP Journal on Embedded Systems (2016) 2016:6
DOI 10.1186/s13639-016-0026-x

EURASIP Journal on Embedded Systems
a SpringerOpen Journal

RESEARCH Open Access

Profit-oriented task scheduling algorithm in Hadoop cluster

Xu-qing Chai[1*], Yong-liang Dong[2] and Jun-fei Li[3]

Abstract

Nowadays, many enterprises provide cloud services based on their own Hadoop clusters. Because the resources of a Hadoop cluster are limited, the Hadoop cluster must select some specific tasks to allocate limited resources in order to get the maximal profit. In this paper, we study the maximal profit problem for a given candidate task set. We describe the candidate task set with a valid sequence and propose a sequence-based scheduling strategy. In order to improve the efficiency of finding a valid sequence, we design some pruning strategies and give the corresponding scheduling algorithm. Finally, we propose a timeout handling algorithm when some task runs timeout. Experiments show that the total profit of the proposed algorithm is very close to the ideal maxima and is obviously bigger than related scheduling algorithms under different experimental settings.

Keywords: MapReduce, Scheduling algorithm, Profit, Big data

1 Introduction

With the rapid development of computer networks and sensor networks, data are exponentially increased, especially on the Internet. In order to deal with large-scale data efficiently, a parallel and distributed cluster with good scalability, flexibility, and fault tolerance is needed. The MapReduce architecture [1], proposed by Google, applies a divide and conquer method to deal with data-intensive tasks and is a de facto standard in big data field. The researches on MapReduce have attracted more and more researchers and engineers. In Google, it uses a large-scale cluster running MapReduce and its related techniques, such as GFS [2] and Bigtable [3], to handle hundreds of petabyte data every week. Based on the analyzing results upon these data, it provides a series of services to people around the world, such as searching, Google earth, advertisements, and so on.

Hadoop [4, 5], contributed by Yahoo!, is the opensource implementation of MapReduce and its related techniques. Hadoop is studied extensively in both academia and industry and has been deployed in many enterprises. Currently, a lot of IT enterprises build their Hadoop/MapReduce clusters and provide all kinds of cloud services to customers. While paying only a little money, the customers can use the powerful Hadoop/MapReduce cluster on demand. During this kind of service process, the service details between enterprises and customers are usually described by a service level agreement (SLA) [6, 7]. The SLAs usually include two kinds, pricing for quantity and pricing for effectiveness. The pricing for quantity SLAs charges the customers proportional to the scale of hardware and the service time. The pricing for effectiveness SLAs charges the customers according to the service effectiveness. Taking the spam email detection service [8] for example, the service must be finished in a certain time, so if only the service finishes within the required time, money would be paid.

In this paper, we study how to schedule customers' tasks to maximize the total profit of a Hadoop cluster. In our research, we mainly focus on the timed MapReduce tasks, which are priced for effectiveness of time, i.e., tasks must be finished within the given time. Here, we abstract each task with four parts, i.e., user-defined Map/Reduce functions, time to complete, profit, and penalty, and we try to find a scheduling algorithm that maximizes the total profit of the Hadoop cluster.

The rest of the paper is organized as follows. In Section 1.1, we briefly describe the MapReduce programming environment and review related works about scheduling algorithms in MapReduce/Hadoop. In Section 1.2, we formalize the problem of maximal profit. In Section 1.3, we propose a sequence-based scheduling strategy and present

* Correspondence: cxq@htu.edu.cn
[1]Network Center of Henan Normal University, Xinxiang City, Henan Province 453000, China
Full list of author information is available at the end of the article

a corresponding scheduling algorithm. Experiments and conclusions are given in Sections 1.4 and 2, respectively.

1.1 Background and related works

In this section, we give a short introduction to MapReduce and then review related works about task scheduling in MapReduce.

1.1.1 Background

MapReduce is a popular programming model for data-intensive tasks and has been widely used in many fields [9–14]. Hadoop is an opensource implementation of MapReduce, and a Hadoop cluster can be made up of thousands of commodity computers. The Hadoop cluster runs on top of the Hadoop distributed file system (HDFS). In the HDFS, data are partitioned into many small chunks and each chunk has multiple backup copies. The multiple backup copy mechanism of HDFS makes the running MapReduce tasks fault-tolerant.

Another advantage of Hadoop is that it is easy to program for programmers. Programmers only need to implement the Map and Reduce functions while processing their massive data, and the details of computing, such as data partitioning, fault tolerance, and communication, are executed automatically by the underlying MapReduce framework. The MapReduce framework is illustrated in Fig. 1. In the user-defined Map function, the input is a key-value pair and the output is zero or more key-value pairs. In the group step, the system group key-value pairs with the same key and they are sent to the same Reduce node. In the user-defined Reduce function, the grouped key-value pairs are handled to generate the results. MapReduce tasks usually need several Map/Reduce iterations.

1.1.2 Related works

In MapReduce, there are some general task schedulers, such as FIFO scheduler [15], capacity-based scheduler [16], and fairness-based scheduler [17]. Concerning the specific applications, Sandholm and Lai [18] proposed a scheduling algorithm, which allows users to adjust the required computing resources dynamically according to the importance of MapReduce tasks, Zaharia et al. [19] proposed a scheduling algorithm for heterogeneous cluster environments, and Kwon et al. [20] proposed the Skewtune algorithm for dealing with skewness in the processes of MapReduce tasks.

In addition, there are some scheduling algorithms, which concern the MapReduce tasks to be finished within a given time. Polo et al. [21] proposed a performance-driven task co-scheduling algorithm, which estimates the required finish time for each task and allocates resources prior for the tasks that cannot be completed timely. Kc and Anyanwu [22] proposed a deadline constraint (DC) scheduler, which tries to allocate a fixed number of Map jobs to each task according to the size of tasks and assumes that each task can utilize all job slots in the Reduce step. However, the workload complementary (WC) scheduling mechanism, proposed by Verma et al. [23, 24], tries to allocate a fixed number of both Map and Reduce jobs to each task according to the size of tasks and to minimize the number of job slots for each task.

1.2 Problem statement

In this paper, we aim to maximize the total profit of a homogeneous Hadoop cluster, where the computing abilities of all nodes are the same. In a Hadoop cluster with M Map jobs and M Reduce jobs, for each submitted task j, we assume the following parameters:

- $j.N_m$, the number of Map jobs in j.
- $j.N_r$, the number of Reduce jobs in j. In order to get high efficiency, both $j.N_m$ and $j.N_r$ are the integer multiples of M.
- j.deadline, the required time or deadline of j.
- j.profit, the profit of j if finished before deadline. Here, we must note that if j does not finish before deadline, then the penalty of j is $j.\text{profit}.\alpha$.

When a lot of customers submit their tasks to a Hadoop cluster at the same time, these tasks form a

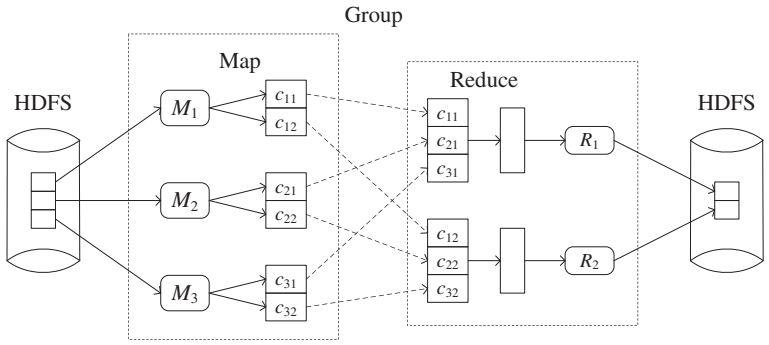

Fig. 1 The MapReduce framework

candidate task set $J = \{j_1, j_2, \ldots, j_{|J|}\}$. The Hadoop cluster needs to select an acceptable task set $J = \left\{j_1^{'}, j_2^{'}, \ldots, j_{|J|}^{'}\right\}$ from J and schedules the selected tasks via a suitable algorithm to finish them. For each task $j_i^{'} \in A$, if $j_i^{'}$ finishes before $j_i^{'}$.deadline, then $j_i^{'}$ is effective, and the profit is $j_i^{'}$.profit; if $j_i^{'}$ does not finish before $j_i^{'}$.deadline, then $j_i^{'}$ is not effective, and the penalty is $j_i^{'}$.profit.α, i.e., the profit is $-j_i^{'}$.profit.α. So, the total profit of the Hadoop cluster is

$$P = \sum_{j \in A, E(j)} j.\text{profit} - \sum_{j^{'} \in A, E(j^{'})} j_i^{'}.\text{profit}.\alpha \quad (1)$$

where $E(\cdot)$ indicates whether or not the given task is effective.

1.3 The proposed scheduling algorithm

In this section, we first propose a sequence-based task scheduling strategy and then propose a scheduling algorithm based on that strategy and finally present an approach for handling timeout.

1.3.1 Sequence-based scheduling strategy

For each task $j \in J$, we can estimate its average processing time for Map jobs, $j.T_m$, and its average processing time for Reduce jobs, $j.T_r$. If all task slots are used to process task j, then it needs $TC_m(j) = \lceil j.N_m/M \rceil \times j.T_m$ time to finish all Map jobs and needs $TC_r(j) = \lceil j.N_r/M \rceil \times j.T_r$ time to finish all Reduce jobs.

1.3.1.1 Definition 1. Sequence For a task set JS (the number of tasks is $|JS|$), a sequence S is a permutation of all tasks in JS, and it specifies the order or jobs according to their finished times. If the finished time of j in the Map step is $COT_m(j)$, then for a given sequence $S = \{j_1, j_2, \ldots, j_{|JS|}\}$, let $COT_m(j_i) < COT_m(j_{i+1})$ for $\forall j_i \in S (0 < i < |JS|)$.

Based on a given sequence S, we propose a scheduling strategy as follows:

- **Map**: When an idle task slot requires a Map job, select a Map job of the first task in sequence S. When all Map jobs of the first task in S are allocated, remove the first task from S.
- **Reduce**: Sort the tasks in JS increasingly according to their times to finish and then get a sorted queue $L_a = \left\{j_1^{'}, j_2^{'}, \ldots, j_{|JS|}^{'}\right\}$. When an idle task slot requires a Reduce job, search in L_d orderly for a task whose Map jobs have all finished and then select a Reduce job of the selected task.

According to the above scheduling strategy, for a given sequence S, we can compute the finished time of the Map step, $COT_m(j)$, and the finished time of the Reduce step, $COT_r(j)$, for any $COT_m(j)$. The computation of $COT_m(j)$ and $COT_r(j)$ is as follows:

- Given a sequence $S = \{j_1, j_2, \ldots, j_{|JS|}\}$, for $\forall j_i \in S$, $COT_m(j_i) = COT_m(j_{i-1}) + TC_m(j) = \sum_{k \in [1,i]} TC_m(j_k)$.
- Given JS and L_d, for the first task $j_1^{'}$ in L_d, its finished time of the Map step can be calculated out, i.e., $COT_m\left(j_1^{'}\right)$, using the above method, then $COT_r\left(j_1^{'}\right) = COT_m\left(j_1^{'}\right) + TC_r\left(j_1^{'}\right)$. We tag the time slice $\left[COT_m\left(j_1^{'}\right), COT_r\left(j_1^{'}\right)\right]$ as occupied. For the ith task in L_d, we first computed $COT_m\left(j_i^{'}\right)$ and then find a series of unoccupied time slices, whose sum to be $TC_r\left(j_i^{'}\right)$, beginning with the moment $COT_m\left(j_i^{'}\right)$, and tag these time slices as occupied. Then, the finished time of the Reduce step of $j_i^{'}$, i.e., $COT_r\left(j_i^{'}\right)$, is the finished time of the latest time slices.

Based on the above scheduling strategy and the computation of finished time, we give the definition of a valid sequence.

1.3.1.2 Definition 2. Valid sequence Given a task set JS and for any sequence S, if for $\forall j_i \in JS$, we have $COT_r(j) \le j.$deadline based on the above scheduling strategy; then, S is a valid sequence.

1.3.1.3 Theorem 1 For a task set JS and a sequence S, the proposed scheduling strategy can make sure that, for $\forall j_i \in JS$, j_i can finish its Map step with minimal time under the constraint of S.

Proof. Given the sequence $S = \{j_1, j_2, \ldots, j_{|JS|}\}$ and its ith task j_i, S ensures that the Map step of j_i starts after all Map steps of $j_k (1 \le k < i)$ finish, i.e., the earliest finished time of the Map step of j_i is $\sum_{k \in [1,i]} TC_m(j_k)$. At the same time, allocating jobs based on the proposed scheduling strategy, the finished time of the Map step of j_i, $COT_m(j_i)$, also equals to $\sum_{k \in [1,i]} TC_m(j_k)$. For $\forall j_i \in S$, we can have the same conclusion.

1.3.1.4 Theorem 2 For a task set JS and a sequence S, if task timeout occurs when using the proposed scheduling strategy, then whatever scheduling strategy is used, it is impossible to finish all tasks in JS on time, and thus, S must be not a valid sequence.

Proof. From theorem 1, we know that the proposed scheduling strategy is optimal in the Map step. Here, we only consider the Reduce step. Assuming that j is a timeout task based on the proposed scheduling strategy, it can be classified into two situations:

- $COT_m(j) + TC_r(j) > j.$ If the Reduce step of j run immediately when its Map jobs finish and still cannot finish on time, then whatever scheduling strategy is used, j cannot finish on time.

- $COT_m(j) + TC_r(j) \le j.$ deadline and $COT_r(j) > j.$ deadline. The finished time of the Reduce step of j is later than deadline. According to the proposed scheduling strategy, there must be some period in time slice $[COT_m(j_i), COT_r(j_i)]$ occupied by other tasks' Reduce jobs, whose time to finish is less than $j.$ deadline. Select the task with minimal time required to finish its Map step and denote it as j'. For all tasks, whose Reduce steps run in time slice $\left[COT_m\left(j_1^{'}\right), COT_r\left(j_1^{'}\right)\right]$, judge whether or not the tasks, whose time to finish is less than $j.$ deadline, exist. If the tasks exist, repeat the above progress until we find a final task j_f, such that all tasks, whose Reduce step runs in time slice $[COT_m(j_f), COT_r(j)]$, and finish later than $j.$ deadline. Obviously, there is no any idle time slice in time slice $[COT_m(j_f), COT_r(j)]$. So, if we use other scheduling strategies to make j finish on time, then there must be some other tasks that will be timeout.

In both of the above situations, it is impossible to finish all tasks in JS on time, so S must be not a valid sequence.

Based on theorems 1 and 2, we can conclude that the proposed scheduling strategy is optimal for a fixed sequence S. That means if timeout tasks under the proposed strategy exist, then they must exist in any other scheduling strategy.

1.3.2 Scheduling algorithm

Based on the proposed sequence-based scheduling strategy, we propose a scheduling algorithm. Firstly, when the candidate task set is static, we use a scoring strategy to specify priorities for all tasks, apply an efficient pruning strategy to find the set of acceptable tasks, and then find a valid sequence. Secondly, when the candidate task set is updated dynamically, we implement an incremental method for judging the set of acceptable tasks and update the valid sequence when necessary.

For a candidate task set c, we need to find the set of acceptable tasks $A = \left\{j_1^{'}, j_2^{'}, \ldots, j_{|A|}^{'}\right\}$, ascertain the valid sequence of A, and then maximize the total profit. However, there are $2^{|J|}$ different acceptable sets for J, and for an acceptable set $A = \left\{j_1^{'}, j_2^{'}, \ldots, j_{|A|}^{'}\right\}$, there are still $|A|!$ different sequences.

In order to improve the efficiency of judging an acceptable set, we first sort all tasks in J and then determine the priority for each task. Upon the features of MapReduce tasks, we mainly consider the following two aspects:

- As the ability of Hadoop cluster is limited, in order to maximize the total profit, the tasks with a bigger profit ratio should be accepted prior. For $\forall j \in J$, the consumed time of the system can be quantified as $STC(j) = TC_m(j) \cdot M/(M+R) + TC_r(j) \cdot R/(M+R)$ and then the profit ratio of j is $P_r(j) = j.\text{profit}/STC(j)$, i.e., the profit per second when running j.
- In MapReduce, if some task j is too long, then most task slots will be idle when running Map/Reduce jobs of j. This would waste lots of resources and affect the accept of other tasks.

Based on the above aspects, we propose a scoring function aiming to maximize the total profit. For a task j, the score is

$$\text{Score}\ (j) = \frac{j.\text{profit}}{STC(j)\cdot \text{Ad}(j)} \tag{2}$$

where $\text{Ad}(j)$ is the adjusting coefficient of j and $STC(j) \cdot \text{Ad}(j)$ is the adjusting time of j. By Eq. 2, the task with a higher score would be given a higher priority.

Let the total consumed time of Map jobs for all $j \in J$ be Total $TC_m = \sum_{j \in J} TC_m(j)$. For task j, compute the average consumed time of all Map jobs for other tasks, $\bar{T}\bar{C}_m(j) = (\text{Total}\ TC_m - TC_m(j))/(|J|-1)$. Given a penalty threshold $\beta(\beta > 1)$, if $TC_m(j) > \bar{T}\bar{C}_m \cdot \beta$, then we think that the Map step of j is too long, and with the same reason, if $TC_r(j) > \bar{T}\bar{C}_r \cdot \beta$, then we think that the Reduce step of j is too long. The computation of the adjusting coefficient $\text{Ad}(j)$ for task j is as follows:

If $TC_m(j) > \bar{T}\bar{C}_m(j)\cdot\beta$, $TC_r(j) > \bar{T}\bar{C}_r(j)\cdot\beta$,
then $\frac{TC_m(j)-\bar{T}\bar{C}_m(j)\cdot\beta}{TC_m(j)} \cdot \frac{M}{M+R} + \frac{TC_r(j)-\bar{T}\bar{C}_r(j)\cdot\beta}{TC_r(j)} \cdot \frac{M}{M+R} + 1;$
if $TC_m(j) > \bar{T}\bar{C}_m(j)\cdot\beta$, $TC_r(j) \le \bar{T}\bar{C}_r(j)\cdot\beta$,
then $\frac{TC_m(j)-\bar{T}\bar{C}_m(j)\cdot\beta}{TC_m(j)} \cdot \frac{M}{M+R} + 1;$
if $TC_m(j) \le \bar{T}\bar{C}_m(j)\cdot\beta$, $TC_r(j) > \bar{T}\bar{C}_r(j)\cdot\beta$,
then $\frac{TC_r(j)-\bar{T}\bar{C}_r(j)\cdot\beta}{TC_r(j)} \cdot \frac{M}{M+R} + 1;$
and if $TC_m(j) \le \bar{T}\bar{C}_m(j)\cdot\beta$, $TC_r(j) \le \bar{T}\bar{C}_r(j)\cdot\beta$,
then $Ad(j) = 1.$

Now, we analyze how to improve the efficiency of finding a valid sequence. Assuming that the candidate set is sorted by Eq. 2, i.e., $\forall j \in J$, Score $(j_i) \le$ Score (j_{i+1}). The brute force searching method needs $(|A|+1)!$ complexity to traverse all candidate sequences. In order to improve the searching speed, we give the following two approaches.

1.3.2.1 Theorem 3 Given a task set A and one of its valid sequences $S = \{j_1, j_2, \ldots, j_n\}$, for a new task j_{new}, there are $n+1$ locations that can be inserted by j_{new}. If $TC_m(j_{\text{new}}) + COT_m(j_i) + TC_r(j_i) > j_i.$ deadline, then j_{new} cannot be inserted into locations $[1, i]$.

Proof. Obviously, if j_{new} is inserted into any location of $[1, i]$, then j_{new} will be timeout.

1.3.2.2 Theorem 4 Given a task set A and one of its valid sequences $S = \{j_1, j_2, \ldots, j_n\}$, for a new task j_{new}, if

$TC_m(j_{\text{new}}) + COT_m(j_i) + TC_r(j_{\text{new}}) > j_i.$ deadline, then j_{new} cannot be inserted into locations $[i+1, n+1]$.

Proof. Assuming that j_{new} can be inserted into one location in $[i+1, n+1]$, according to the proposed scheduling strategy, we have that the earliest finished time of j_{new} is equal to or larger than $TC_m(j_{\text{new}}) + COT_m(j_i) + TC_r(j_{\text{new}})$. So, j_{new} must be timeout.

Based on theorems 3 and 4, we proposed an algorithm for rapidly finding the acceptable set and its corresponding valid sequence, and the details are in algorithm 1. With the proposed algorithm, the maximum profit for the candidate task set can be got.

Input: Submitted job set $J = \{j_1, j_2, \cdots, j_{|J|}\}$
Output: Receive job set A and its valid sequence ϕ
Computer the rank score for each job in J via equation 2;
Sort jobs in J decreasingly according to their scores;
Let $A \leftarrow \{j_1\}, \phi_1 \leftarrow \{\{j_1\}\}$
For each $j_i \in J$ and $i \neq 1$ do
  Let $\phi_i \leftarrow \{\}$ for each $S \in \phi_{i-1}$ do
    Judge the minimal location a that j_i can be inserted
    Via theorem 3;
    If $a \leq b$ then
      For each $p \in [a, b]$ do
        insert j_i into S at location p and get S'
        Compute the required finish time for all jobs
        In S' with SEQ strategy;
        If S' is a valid sequence then
          Let $\phi_i \leftarrow \phi_i \cup S'$
        If $\phi \neq \varphi$ then
          Let $A \leftarrow A \cup j_i$
        else
          Drop j_i and let $\phi_i \leftarrow \phi_{i-1}$
Return A and any valid sequence in $\phi_{|J|}$

1.3.3 Timeout handling approach

We propose the above scheduling algorithm in the homogeneous Hadoop cluster, and in most cases, the estimation values of $j.T_m$ and $j.T_r$ are close to real values. However, in some abnormal situations, such as network congestion and node crashes, some accepted tasks cannot finish on time. In these situations, we must adjust the running tasks in order to get the maximum profit.

According to Eq. 1, in order to get the maximum profit, we should drop the task with the lowest profit while making other tasks finish on time. Based on this idea, we propose a timeout handling algorithm, and the details of the algorithm are in algorithm 2.

Input: Receive job set A ,valid sequence S and running job $j_{running}$
Output: the dropped set C
Sort jobs in A increasingly according to their profits;
If $j_{running}$ equeals to the job with minimal profit then
  $C = j_{running}$
else
  For each $j_i \in A$ and $j_i.profit < j_{running}.prifit$ do
    Remove j_i form S
    If S is a valid sequence then
      Let $C \leftarrow \{j_i\}$;
      Break;
    Else if
    $j_i.profit + \sum_{j \in C} j.prifit < j_{running}.profit$ then
      Let $C \leftarrow C \cup \{j_i\}$
      If $S - C$ is a valid sequence then
        Break;
      else
        $C = \{j_{running}\}$

1.4 Experiments

1.4.1 Experimental setting

In the experiments, the Hadoop cluster contains one master node and 40 slave nodes, and each node contains an Intel Core i3 3.1 GHz CPU, 8 GB memory, and 500 GB storage and runs Redhat Linux 6.1. In the slave nodes, each node is configured with two Map task slots and two Reduce task slots.

The dataset we use in the experiments is the enwiki (https://dumps.wikimedia.org/enwiki/20150204/), and we run three classical tasks on the dataset, i.e., statistics of word frequencies, inverted index, and distributed grep. The dataset is stored on the Hadoop file system (HDFS), each chunk is 64 MB, and each data chunk has three copies. For a candidate task set *J*, we mainly consider the following three parameters that affect the performance:

- Average task size *L*, i.e., the average size (number of chunks) of all tasks in *L*;
- Task number *N*, i.e., the number of tasks in *L*;
- Average deadline *D*, i.e., the average deadline (time to finish) of all tasks in *L*.

The computation of total profit is in Eq. 1. In addition, we define receive rate and finish rate as follows:

$$\begin{aligned} \text{Receive rate} &= \frac{\text{Size of received task set}}{\text{Size of candidate task set}} \\ \text{Finish rate} &= \frac{\text{Number of finished tasks}}{\text{Number of accepted tasks}} \end{aligned} \tag{3}$$

1.4.2 Results

The baseline algorithms we use in the experiments are DC [22] and WC [24].

Firstly, we evaluate the effect of task number on the total profit, and the results are in Fig. 2. In Fig. 2a, the ideal curve is the ideal profit, and with the increasing of average task size, all profit values decrease, but our proposed approach is close to the ideal value. In Fig. 2b, all of the three receive rates decrease gradually, but our approach has the highest value, which means that our approach can receive the most candidate tasks. In

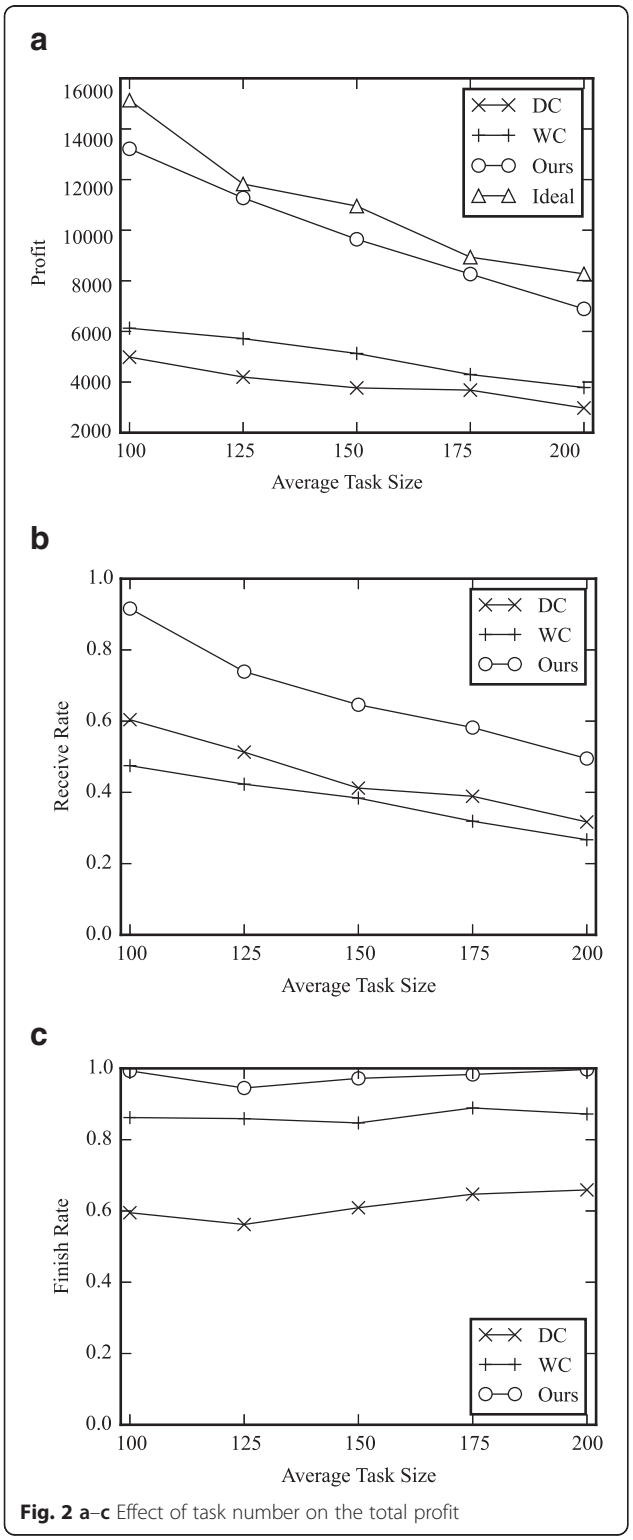

Fig. 2 a–c Effect of task number on the total profit

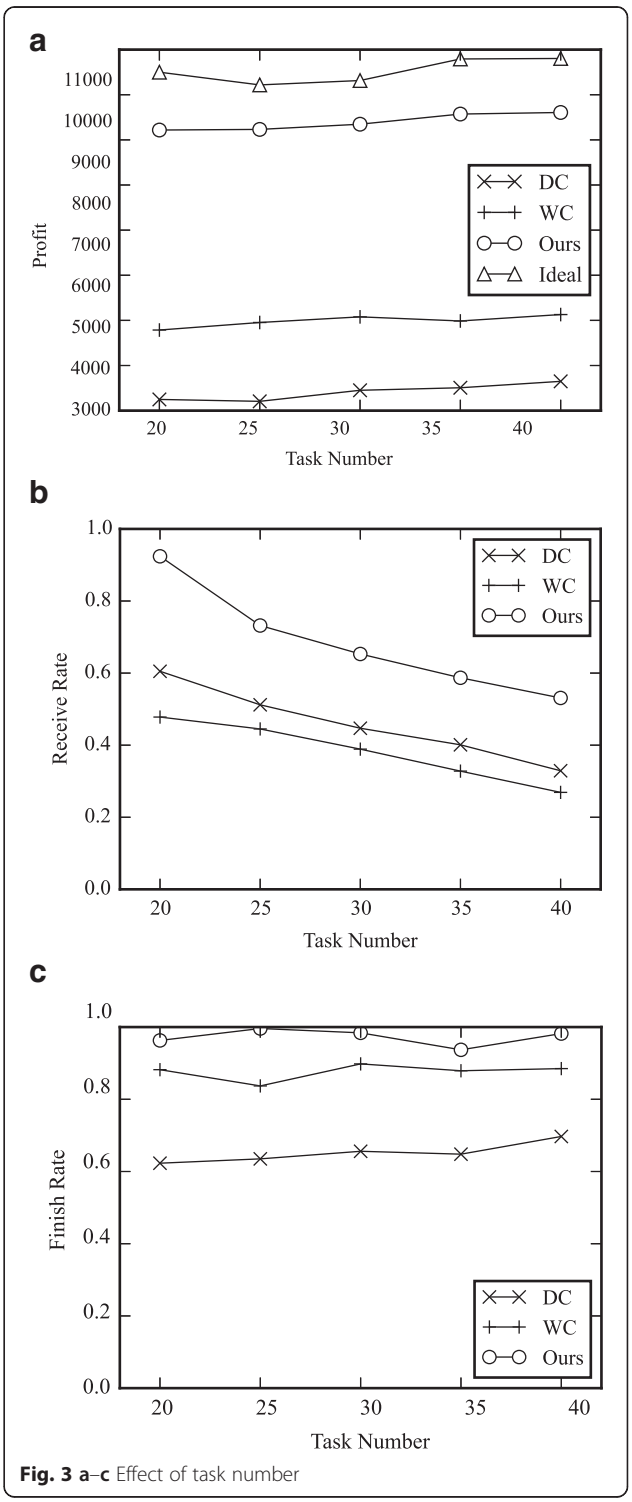

Fig. 3 a–c Effect of task number

Fig. 2c, the proposed approach has a much higher finish rate than the other two. As our approach not only receives the most candidate tasks but also finishes most of them, so it can bring the most total profit.

Meanwhile, we observe the effects of task number and average deadline on the total profit, and the results are shown in Figs. 3 and 4, respectively. With the same reason, our approach not only receives the most candidate tasks but also finishes most of them, so it can bring the most total profit. In addition, the total profits of our approach for three situations are very close the ideal values.

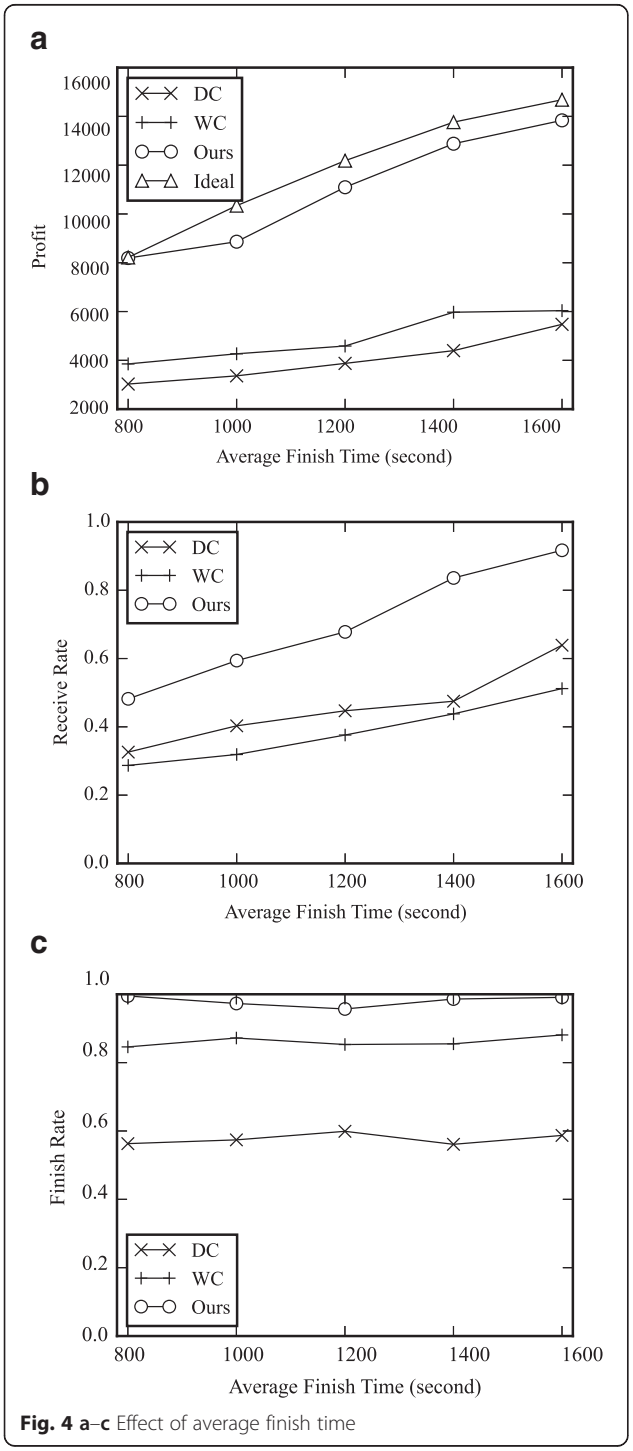

Fig. 4 a–c Effect of average finish time

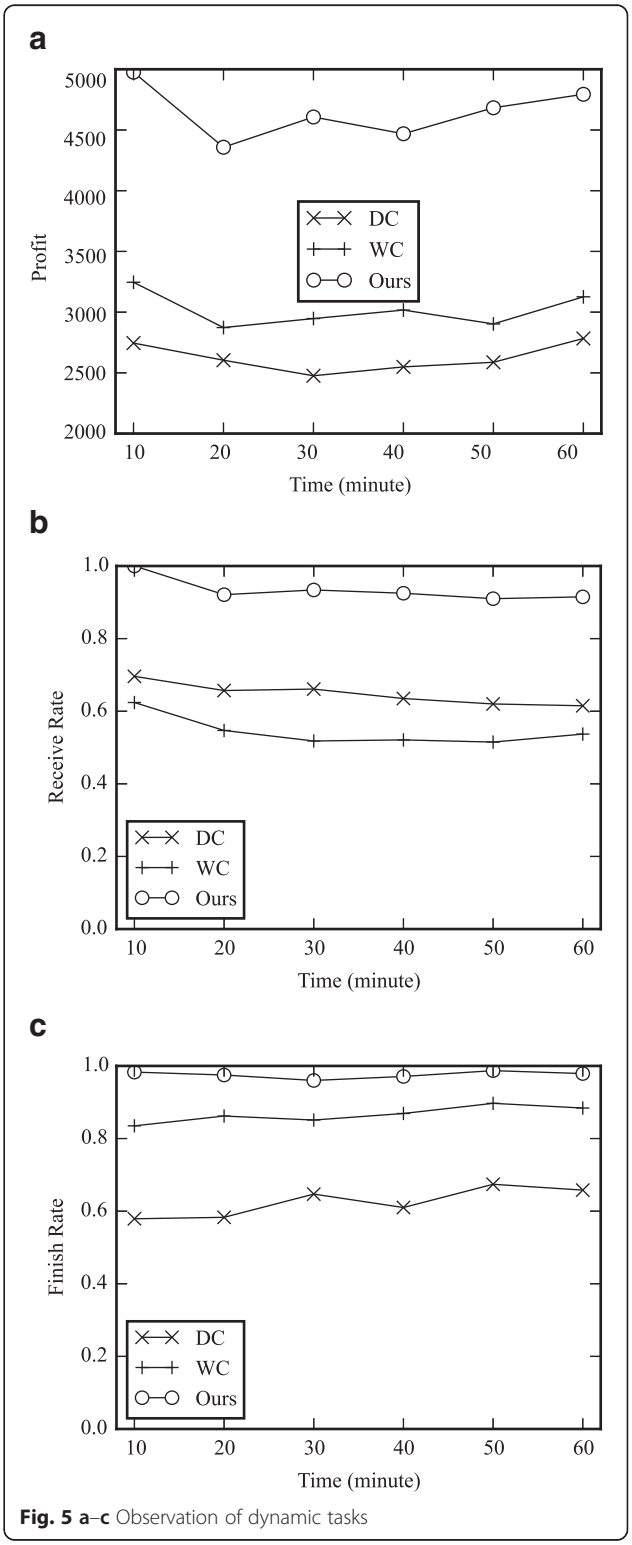

Fig. 5 a–c Observation of dynamic tasks

Finally, we dynamically submit the tasks to the Hadoop cluster and observe the changes of the total profit. In Fig. 5, the horizontal axis is the elapsed time and the vertical axes are the total profit, receive rate, and finish rate, respectively. As we can see from the figure, our approach not only receives the most candidate tasks but also finishes most of them, so it can bring the most total profit. This illustrates that the proposed approach is also suitable to tasks that are submitted dynamically.

2 Conclusions

In this paper, we study the problem of maximal profit in a Hadoop cluster, where the resources are not enough for the whole candidate task set. In order to maximize the total profit, we select some high-profit ratio tasks based on the valid sequence of a candidate task set. Furthermore, in order to improve the efficiency of finding a valid sequence, we design some pruning strategies and give the corresponding scheduling algorithm. We also propose a timeout handling algorithm. Experiments show that the total profit of the proposed algorithm is very close to the ideal maxima and is obviously bigger than related scheduling algorithms under different experimental settings.

Competing interests
The authors declare that they have no competing interests.

Acknowledgements
This work was supported by the following funds: Department of Science and Technology of Henan (9412012Y0004, 9412012Y0005) and Education Department of Henan (13A510520, 2013-gh-12, 14A520053, SKL-2014-795).

Author details
[1]Network Center of Henan Normal University, Xinxiang City, Henan Province 453000, China. [2]XinLian Collage of Henan Normal University, zhengzhou City, Henan Province 453000, China. [3]Shanghai Jing Sheng Communication Technology Co Ltd, Shanghai City 021, China.

Received: 25 November 2015 Accepted: 24 February 2016
Published online: 29 March 2016

References

1. J Dean, S Ghemawat, MapReduce: simplified data processing on large clusters. Commun ACM **51**(1), 107–113 (2008)
2. S. Ghemawat, H. Gobioff, S.-T. Leung, in *ACM SIGOPS Operating Systems Review*. The Google file system, vol. 37 (ACM, 2003), pp. 29–43
3. F Chang, J Dean, S Ghemawat, WC Hsieh, DA Wallach, M Burrows, T Chandra, A Fikes, RE Gruber, Bigtable: a distributed storage system for structured data. ACM Transactions on Computer Systems (TOCS) **26**(2), 4 (2008)
4. D Borthakur, The Hadoop distributed file system: architecture and design. Hadoop Project Website **11**(2007), 21 (2007)
5. K. Shvachko, H. Kuang, S. Radia, R. Chansler, in *Mass Storage Systems and Technologies* (MSST). The Hadoop distributed file system. 2010 IEEE 26th Symposium On, (IEEE, 2010), pp. 1–10.
6. J.M. Peha, F. Tobagi et al., in *INFOCOM'91. Proceedings. Tenth Annual Joint Conference of the IEEE Computer and Communications Societies. Networking in the 90s*. A cost-based scheduling algorithm to support integrated services, (IEEE, 1991), pp. 741–753.
7. Y Chi, HJ Moon, H Hacigümüs, iCBS: incremental cost-based scheduling under piecewise linear SLAs. Proceedings of the VLDB Endowment **4**(9), 563–574 (2011)
8. M.T.B. Aun, B.-M. Goi, V.T.H. Kim, in *Sustainable Utilization and Development in Engineering and Technology (STUDENT), 2011 IEEE Conference On*. Cloud enabled spam filtering services: challenges and opportunities, (IEEE, 2011), pp. 63–68.
9. A McKenna, M Hanna, E Banks, A Sivachenko, K Cibulskis, A Kernytsky, K Garimella, D Altshuler, S Gabriel, M Daly et al., The genome analysis toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. Genome Res **20**(9), 1297–1303 (2010)
10. R.L. Ferreira Cordeiro, C. Traina Junior, A.J. Machado Traina, J. López, U. Kang, C. Faloutsos, in *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Clustering very large multi-dimensional datasets with MapReduce, (ACM, 2011), pp. 690–698.
11. K Wiley, A Connolly, J Gardner, S Krughoff, M Balazinska, B Howe, Y Kwon, Y Bu, Astronomy in the cloud: using MapReduce for image co-addition. Stronomy **123**(901), 366–380 (2011)
12. M.F. Husain, P. Doshi, L. Khan, B. Thuraisingham, in *Cloud Computing*. Storage and retrieval of large RDF graph using Hadoop and MapReduce, (Springer, 2009), pp. 680–686.
13. W Dou, X Zhang, J Chen, KASR: a keyword-aware service recommendation method on MapReduce for big data application. IEEE Transactions on Parallel & Distributed Systems **1**, 1 (2014)
14. D Dahiphale, R Karve, AV Vasilakos, H Liu, Z Yu, A Chhajer, J Wang, C Wang, An advanced MapReduce: cloud MapReduce, enhancements and applications. Network and Service Management, IEEE Transactions on **11**(1), 101–115 (2014)
15. RB Thirumala, Survey on improved scheduling in Hadoop MapReduce in cloud environments. Int J Comput Appl **34**(9), 29–33 (2011)
16. M. Yong, N. Garegrat, S. Mohan, in *Proceedings of the 2009 IEEE International Conference on Web Services*. Towards a resource aware scheduler in Hadoop, (Los Angeles, CA, USA, 2009), pp. 102–109
17. M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, I. Stoica, in *Proceedings of the 5th European Conference on Computer Systems*. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling, (ACM, 2010), pp. 265–278.
18. T. Sandholm, K. Lai, in *Job Scheduling Strategies for Parallel Processing*. Dynamic proportional share scheduling in Hadoop, (Springer, 2010), pp. 110–131.
19. M. Zaharia, A. Konwinski, A.D. Joseph, R.H. Katz, I. Stoica, in *OSDI*. Improving MapReduce performance in heterogeneous environments, vol. 8 (2008), p. 7
20. Y. Kwon, M. Balazinska, B. Howe, J. Rolia, in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. Skewtune: mitigating skew in MapReduce applications, (ACM, 2012), pp. 25–36.
21. J. Polo, D. Carrera, Y. Becerra, J. Torres, E. Ayguadé, M. Steinder, I. Whalley, in *Network Operations and Management Symposium (NOMS)*. Performance-driven task co-scheduling for MapReduce environments, (IEEE, 2010), pp. 373–380.
22. K. Kc, K. Anyanwu, in *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference On*. Scheduling Hadoop jobs to meet deadlines, (IEEE, 2010), pp. 388–392.
23. A. Verma, L. Cherkasova, R.H. Campbell, in *Proceedings of the 8th ACM International Conference on Autonomic Computing*. Aria: automatic resource inference and allocation for MapReduce environments, (ACM, 2011), pp. 235–244.
24. A. Verma, L. Cherkasova, V.S. Kumar, R.H. Campbell, in *Network Operations an d Management Symposium (NOMS)*. Deadline-based workload management for MapReduce environments: Pieces of the performance puzzle, (IEEE, 2012), pp. 900–905.